ECE371 Neural Networks and Deep Learning Assignment 1

lixh278 and 22308089
School of Electronics and Communication Engineering
Sun Yat-sen University, Shenzhen Campus
lixh278@mail2.sysu.edu.cn

Abstract: This assignment focuses on the development of a deep learning model for flower image classification using PyTorch. The model leverages data augmentation, transfer learning, and optimization techniques to achieve high accuracy. The main text should not exceed 4 pages, excluding references.

Keywords: Deep Learning, Image Classification, Transfer Learning, PyTorch

1 Introduction

The purpose of this assignment is to develop a deep learning model for classifying flower images. The task involves preparing a dataset, designing a neural network architecture, training the model, and evaluating its performance. This report outlines the methodology, experiments, and findings of the project. The assignment report must be submitted electronically via GitHub Classroom. Please ensure you have a GitHub account (https://github.com) set up in advance. We will provide the assignment link at a later time.

2 Related Work

Image classification is a fundamental task in computer vision, with numerous applications in various fields. Recent advancements in deep learning have significantly improved the performance of image classification models. Convolutional Neural Networks (CNNs) have emerged as the state-of-the-art architecture for this task. Transfer learning, which involves using pre-trained models on large datasets like ImageNet, has proven to be highly effective in improving the accuracy and reducing the training time for specific tasks. Techniques such as data augmentation, learning rate scheduling, and optimization algorithms like Adam have also played crucial roles in enhancing model performance. For example, **?**] demonstrated the effectiveness of deep CNNs for image classification, and **?**] introduced the ResNet architecture, which has become a cornerstone in many deep learning applications.

3 Method

This section details the implementation of the flower image classification model using PyTorch. The methodology includes data preparation, model architecture, loss function, optimizer, and training process.

3.1 Data Preparation

The dataset consists of flower images categorized into different classes. Data augmentation techniques such as random cropping, horizontal flipping, rotation, color jittering, and affine transformations were applied to enhance the model's robustness. The dataset was split into training and validation sets, with 80% for training and 20% for validation. Data normalization was performed using the mean and standard deviation values from the ImageNet dataset.

3.2 Model Architecture

A pre-trained ResNet18 model was used as the base architecture. The final fully connected layer was modified to match the number of flower classes. This approach leverages the feature extraction capabilities of the pre-trained model while adapting it to the specific classification task.

3.3 Loss Function and Optimizer

The cross-entropy loss function was used to measure the difference between the predicted and actual labels. The Adam optimizer was chosen for its efficiency and adaptive learning rate properties. A learning rate scheduler was implemented to adjust the learning rate during training, improving convergence.

3.4 Training Process

The training process involved iterating over the training dataset in batches, performing forward and backward passes, and updating the model parameters. The model's performance was evaluated on the validation set after each epoch. The best model was selected based on the highest validation accuracy, and its weights were saved for future use.

4 Experiments

This section presents the experimental results, including validation accuracy, loss curves, and a discussion of the findings.

4.1 Experimental Setup

The experiments were conducted using a GPU-enabled environment to accelerate training. The model was trained for 25 epochs with a batch size of 32. The learning rate was initially set to 0.001 and adjusted using a step learning rate scheduler. For the second experiment, the learning rate was further reduced to 0.0001, and the model was trained for an additional 25 epochs, totaling 50 epochs.

4.2 Results

- **Experiment 1 (Epochs: 25, Learning Rate: 0.001)**:
  - Training complete in 3 minutes 20 seconds.
  - Best validation accuracy: 89.82%.
- **Experiment 2 (Epochs: 50, Learning Rate: 0.0001)**:
  - Training complete in 6 minutes 49 seconds.
  - Best validation accuracy: 93.33%.

4.3 Discussion

The results demonstrate the effectiveness of the chosen architecture and training techniques. Data augmentation and transfer learning significantly contributed to the model's performance. The initial learning rate of 0.001 allowed the model to converge quickly, while the reduced learning rate of 0.0001 in the second experiment further fine-tuned the model, resulting in a higher validation accuracy. However, overfitting was observed in the later stages of training, suggesting the need for further regularization techniques such as dropout or early stopping. Future work could explore more advanced architectures and hyperparameter tuning to further improve the model's accuracy.

References